Design of a Multi-Strategy Parallelization for an Entire Application of Document Categorization on Low-Cost Multiprocessor PCs

Parallélisations de catégorisation de document sur multiprocesseur

S. Vialle — G. Schaeffer — M. Ianotto

ABSTRACT. This paper introduces a research about parallelization of an entire application of Document-Categorization. The objective of this parallel computing research is to obtain a parallelization that can be successfully used on low cost and largely diffused shared memory multiprocessor PCs (not only on powerful and expensive supercomputers), and without any change in the input, output and user interface of the application (under Windows OS). This is a first step toward a parallelization on a cluster of multiprocessor PC, a more generic and still low cost parallel architecture. In this article, we describe parallel algorithms and programming technics we have designed to reach good performances on low cost but limited PC architecture. This leads us to introduce different parallelization strategies, for the different parts of the application, dealing with numerous disk accesses and the variety of configurations chosen by the users. Each parallelization is described and evaluated, and global performances of the final mix are introduced on 4-processor PC with SCSI disk technology and on a more recent 2-processor PC with IDE disk technology, leading to different but significant decreases of execution time. Then we can upgrade regularly our parallel machines to remain competitive compared to new sequential machines, because their low cost allows frequent upgrade and we always reach interesting speed up.

The chosen application has been first designed to easily evaluate some classification algorithms (useful to Text-Mining researchers), and second to detect errors in previous manually categorizations and to advise some changes (useful to end-users).

RÉSUMÉ. Cet article introduit une recherche sur la parallélisation d’une application complète de catégorisation de documents. L’objectif est d’obtenir une parallélisation pouvant être utilisée de manière efficace sur un multiprocesseur bon marché, largement répandu.

Studia Informatica Universalis.
à mémoire partagée PC, sans changement dans les données d’entrée, ni les sorties et utilisant l’interface de l’application sous Windows OS. Il s’agit d’une première étape vers une parallélisation sur une grappe de multiprocesseurs PC, architecture parallèle la plus générique et encore peu chère. Nous décrivons les algorithmes parallèles et les techniques de programmation que nous avons mis au point pour atteindre de bonnes performances sur cette architecture PC bon marché mais limitée. Ceci nous a conduit à introduire différentes stratégies de parallélisation en fonction des parties de l’application, négociant avec les accès disques et la variété de configurations susceptibles d’être choisies par les utilisateurs. Chaque parallélisation est explicitée et évaluée ainsi que les performances sur des PC quadri-processeurs avec la technologie de disques SCSI, ainsi que sur un PC plus récent bi-processeur avec technologie IDE pour les disques. Ainsi nous pouvons mettre-à-jour régulièrement notre système parallèle pour rester compétitif par rapport aux nouvelles machines séquentielles, à des prix bas permettant de fréquentes mises-à-jour tout en atteignant des accélérations intéressantes. L’application choisie a d’abord été conçue pour l’évaluation d’algorithmes de classification utilisés par les chercheurs en fouille de textes et ensuite pour détecter les erreurs dans les catégorisations manuelles précédentes, ainsi que pour conseiller des changements utiles aux utilisateurs finaux.

**KEYWORDS**: Parallelization, clusters, text mining, classification

**MOTS-CLÉS** : Parallélisation, grappes de processeurs, architecture, fouille de textes, classification
This paper introduces a research about parallelization of an entire application of Document-Categorization. The objective of this parallel computing research is to obtain a parallelization that can be successfully used on low cost and largely diffused shared memory multiprocessor PCs (not only on powerful and expensive supercomputers), and without any change in the input, output and user interface of the application (under Windows OS). This is a first step toward a parallelization on a cluster of multiprocessor PC, a more generic and still low cost parallel architecture.

In this article, we describe parallel algorithms and programming techniques we have designed to reach good performances on low cost but limited PC architecture. This leads us to introduce different parallelization strategies, for the different parts of the application, dealing with numerous disk accesses and the variety of configurations chosen by the users. Each parallelization is described and evaluated, and global performances of the final mix are introduced on 4-processor PC with SCSI disk technology and on a more recent 2-processor PC with IDE disk technology, leading to different but significant decreases of execution time. Then we can upgrade regularly our parallel machines to remain competitive compared to new sequential machines, because their low cost allows frequent upgrade and we always reach interesting speed up. The chosen application has been first designed to easily evaluate some classification algorithms (useful to Text-Mining researchers), and second to detect errors in previous manually categorizations and to advise some changes (useful to end-users).

1. Motivations and Objectives

Research introduced in this paper is about the parallelization on low cost multiprocessor PCs of an entire application of document categorization. Beyond the interest to design efficient parallel algorithms our motivation has two main foundations:

– First, document categorization is part of Text-Mining: a computing science domain with increasing interest, with many applications on very large databases such as web data. Then it usually needs great amount of resources (CPU, memory and disk space), parallel computing tech-
nics can be helpful (Talia, 2002, Hedberg, 1995) and many operational parallelizations exist in Text-Mining and Data-Mining on large supercomputers (such as CM2, CM5 or IBM SP2) (Freitas et al., 2000).

Second, a great problem of parallel computing is the price of supercomputers. This has led to look for alternative solution based on cheap technology: PC-based parallel computers. One solution is the multiprocessor PC, another is the PC cluster, or the multiprocessor-PC cluster. But standard PC architecture has weaknesses, such as poor sharing memory mechanism and poor concurrent disk accesses, and parallel programming on PC-based architecture can be hard if application is not mainly composed of independent computations.

So, we were interested to design parallel algorithms and to experiment parallel programming for Document-Categorization on PC-based parallel architectures. We have decided to begin with shared memory multiprocessor PC (mainly a four processor PC), as a first step toward clusters of multiprocessor PCs.

We have chosen a complete Document-Categorization application to parallelize, not just a generic algorithm used to classify documents, in order to be in front of many problems encountered in Document-Categorization: document parsing including disk accesses, data analysis including numerous accesses to large memory spaces, user configuration change such as classification algorithm or number of relevant words used to encode documents. Parallel computing will be interesting to Document-Categorization only if it can reach speed up and deal with all these problems. This has led us to design different parallelization strategies for the different parts of the application, and to make a compatible mix in a new application with the same input, output and user interface than the previous sequential application.

Numerous experimentations and parallel performance measurements are introduced in this paper and allow to evaluate our parallelization strategies on a 4-processor PC. Finally, we have experimented our parallel application on different multiprocessor PCs, marketed at different dates, with different processors, to test the portability and the reproducibility of the reached speed up.
2. Short application introduction

Document categorization is a large research area (Yang et al., 1999, Yang et al., 1997), included in Text-Mining domain. As we wanted to experiment a complete parallelization of an entire application, not just parallelization of a generic algorithm, we have focused on only one application.

We have chosen a document categorization application, designed to parse and classify large number of messages, already manually classified in some categories. The initial goal of this application was to evaluate classification algorithms, comparing initial categorization and results of the classification algorithm under test. This is a very useful application, frequently run, by our Text-Mining researcher colleagues, and they wanted a parallel version to make easier their numerous experimentations of classification algorithms. Figure 3.1 shows the global software architecture of this application, with two pre-treatment modules \((m\cdot1\) and \(m\cdot2\)) and a classification module \((m\cdot3)\). But we can add a re-classification module \((m\cdot4)\) to obtain a new application, not for Text-Mining researchers, but to help end-users to check an initial categorization and detect classified messages in unadapted categories. Then \(m\cdot4\) module decides to reclassify these messages or just to point out and to ask for human final decision. For example, this could be useful to improve organization of classified advertisements, when users enter their message in a wrong category, or to detect parasite advertisements in news groups, and move to trash categories.

This application has been developed in our laboratory, initially on sequential PC under Windows (using C++ language), and had long runs on a 20000 message base (20 news groups, 1000 messages each). So, this application was available for parallelization on shared memory multiprocessor PC under Windows, and it was a challenge to do it without any change in its input and output and results. This was necessary to obtain a parallelization interesting for our user-colleagues.

3. Software architecture of the application

Our document categorization application is based on a 3 stage soft-
ware architecture, that allows to experiment and evaluate some classification algorithms, or a 4 stage software architecture if we wish to change the initial classification, according to the results of the new classification algorithm. See the 4 software modules on figure 3.1.

— m-1 module is a pre-treatment stage that filters the input documents to produce the full vocabulary set after removing very low frequency words (irrelevant for classification) and very high frequency words (such as articles). Then this first module counts and stores the number of instances of each word in every message, and builds a full vocabulary of the entire database. This module spends long time in numerous accesses to *ascii* files, and its parallelization on PC architecture encounters some limitations, see section 4.

— m-2 module is a second pre-treatment stage, that creates a vector database from the filtered message database and full vocabulary set. The user has to fix some parameters before to run this module, such as the number of relevant words to characterize each message, and the heuristic to determine these words among the entire database. This module makes important file accesses, as first module, but many accesses are
fast ones on binary files and parallelization reaches good performances when using not too many relevant words to encode documents.

– \( m-3 \) module is the kernel of the application: the classification routine that is under evaluation or used to track initial classification errors. Two algorithms have been implemented and parallelized, see section 6, and parallelization performances depend on the classification algorithm used.

Last stage (\( m-4 \) module) is still sequentially implemented, but is not taken into account in this research because our user-colleagues are mainly interested by the output of the third stage: statistical results of the re-classification and comparison to the initial classification, to evaluate new classification algorithms. So, we have parallelized only the three modules \( m-1, m-2 \) and \( m-3 \).

Before to introduce each module parallelization, we point out that all modules read and write files, and communicate through these files. This mechanism could be improved, using more in-memory storage. But first, this file based strategy is adapted to large database processed by more complete Text-Mining applications when data are larger than memory, and second, it is well adapted to research activity of our user-colleagues that want to save pre-treatment results. So, we consider these many file read and write operations as useful features of the application, that we try to optimize and parallelize but to not remove.

4. Parallelization of the message filtering and vocabulary building

Present course of our categorization first consists in accumulating messages sorted by categories following author decisions, and second in running the categorization application, beginning with the \( m-1 \) pre-treatment module that reads files of each category, one category after the other.

The \( m-1 \) pre-treatment module is composed of three parts, corresponding to three different algorithms. First, it reads each message of each category (News Group (NG)), filters it (suppressing irrelevant words), stores resulting words in a hash-table and concatenates results in just one file per category. This part is the most time consuming of
the module and its parallelization is based on a dynamic load balancing strategy. Second, the $m-1$ module computes average mutual informations (Cover et al., 1991, McCallum et al., 1998), that will be used to identify most relevant words by $m-2$ module. This computational part is limited to approximately 0.4% of the sequential execution time of the module, and it has not been parallelized for the moment, and is always a serial computation. Third, the $m-1$ module saves on disk the full vocabulary database that it has built in the hash-table. This last part has been parallelized using a static data partitioning of the hash-table. Algorithms of these three parts are illustrated on the figure 4.2.
4.1. Parallelization of message processing (first step)

The first step of m-1 module consists in message filtering and vocabulary database building (message processing), see figure 4.2. It runs a processing loop on message categories (or news groups: NG), and two synchronization barriers delimit each call to the NG message processing routine (detailed on figure 4.3). But synchronization barriers were not available in our programming environment. As we wanted to avoid any special parallel tool usage (to increase portability), we have designed and implemented a non-busy-waiting synchronization barrier, based on standard semaphore use. Then, using this home-made barrier, all threads read and filter messages of the same category at the same time.

However, messages have different sizes and need different processing times. We have adopted a dynamic load balancing strategy: each thread gets a message (the next available in the category), processes it, and then tries to get another immediately. Short messages are processed faster, and some threads process more messages than others, but all threads are working while there are messages to process, improving
the load balancing (see figure 4.3 for details). But some performance limitations remain. Message processing routine contains three critical sections: message name get, hash-table building, and category file writing (file containing all filtered messages of one category). Each critical section protects a different resource and then uses a different lock (binary semaphore). Differences in message sizes and then in corresponding processing times, statistically prevent threads to try to enter critical sections at the same time and to waste too many time on mutex locks. But to reduce again waiting times we alternate file writing and hash-table building on odd and even threads. This simple optimization has led to small but sensitive and reproducible execution time decrease (10 second less on 400s run of m-1 module on 4 processors). This is not an important decrease, but alternate two independent operations on odd and even processors is so easy that we have to do it.

This parallelization strategy (of first step of m-1 module) seems to be most adapted to database containing a large number of message per category, than to a large number of categories with few messages each. For example, it is well adapted to classification of a set of very active news groups, that is our typical benchmark.

4.2. Parallelization of vocabulary database save (last step)

When all messages are filtered and all relevant words are entered in the hash-table, we need to save this table in a vocabulary file. To parallelize this last step of m-1 module we have partitioned the hash-table: each thread processes the same number of entries, formats a string per word in the hash-table, and writes this string in the vocabulary file. Of course, these write operations are enclosed in a critical section (see figure 4.2), that introduces waiting times. However, this basic parallelization has led to significant speed up.

4.3. m-1 module global performances

Table 1 contains performance measurements of this first module (m-1), on our medium and on our large databases (1872 and 20000 mes-
Multi-Strategy Parallelization

Medium database: 1872 msg

<table>
<thead>
<tr>
<th>Number of proc.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. time (s)</td>
<td>15.38</td>
<td>8.94</td>
<td>7.14</td>
<td>6.47</td>
</tr>
<tr>
<td>Speed Up</td>
<td>1.0</td>
<td>1.7</td>
<td>2.2</td>
<td>2.4</td>
</tr>
<tr>
<td>Efficiency (%)</td>
<td>100</td>
<td>86</td>
<td>72</td>
<td>60</td>
</tr>
</tbody>
</table>

Large database: 20000 msg

<table>
<thead>
<tr>
<th>Number of proc.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. time (s)</td>
<td>289</td>
<td>171</td>
<td>136</td>
<td>129</td>
</tr>
<tr>
<td>Speed Up</td>
<td>1.0</td>
<td>1.7</td>
<td>2.1</td>
<td>2.2</td>
</tr>
<tr>
<td>Efficiency (%)</td>
<td>100</td>
<td>85</td>
<td>71</td>
<td>56</td>
</tr>
</tbody>
</table>

Table 1: Performance of the entire first module (message filtering and vocabulary file building) on a 4xPPI-450MHz PC with SCSI disk technology

Sot our multi-strategy parallelization, including synchronization barriers and critical sections, dynamic load balancing and data partitioning, with embedded computations and disk accesses, is efficient even on the basic parallel architecture of a 4-processor PC. Each parallelization and optimization described in previous sections has contributed to these results.

5. Parallelization of the vector database building

The $m$-2 module is first in charge to read the full vocabulary file and second has to identify the $N$ most relevant words to classify all the messages, conforming to the user choices of $N$ and of the heuristic to use. Third, $m$-2 module reads all category files of filtered messages (generated by module $m$-1) and computes the vector of each message: counting all instances of relevant words in the message. Next, messages will be represented only by their vectors, that will be saved on disk (last operation of $m$-2 module). Figure 5.4 shows the main steps of the $m$-2 module algorithm. As for $m$-1 module, we have used different strategies.
Figure 5.4: Parallel algorithm of m-2 module: Identification of relevant words, and generation of vector message files
Table 2: **Performance of the entire second module: full vocabulary rebuild and message vector generation and save, on a 4xP11-450MHz PC with SCSI disk technology. Performances are good on large database only for medium number of relevant words used to classify messages** (top table).

<table>
<thead>
<tr>
<th>Number of proc.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. time (s)</td>
<td>30</td>
<td>16</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>Speed Up</td>
<td>1.0</td>
<td>1.9</td>
<td>2.5</td>
<td>2.7</td>
</tr>
<tr>
<td>Efficiency (%)</td>
<td>100</td>
<td>94</td>
<td>83</td>
<td>68</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of proc.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. time (s)</td>
<td>47</td>
<td>43</td>
<td>54</td>
<td>45</td>
</tr>
<tr>
<td>Speed Up</td>
<td>1.0</td>
<td>1.1</td>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>Efficiency (%)</td>
<td>100</td>
<td>55</td>
<td>29</td>
<td>26</td>
</tr>
</tbody>
</table>

5.1. **Parallelization of the full vocabulary rebuild (first step)**

First step of m-2 module reads the vocabulary file and stores words in a hash-table, the reverse of the last operation of m-1 module. Like in m-1 module, the parallelization is based on *data-partitioning*: each thread reads a part of the *ascii* file and enters words in the hash-table.

However PC architecture is not designed to support parallel disk accesses, and hash-table filling is a critical section that needs to be protected by mutex. So it was not obvious to obtain speed up parallelizing this first step of m-2 module, but the vocabulary file to read is an *ascii* file, and *ascii* IO operations are usually long and are good candidates to parallelization. Then we have decided to experiment a parallelization of this first step of m-2 module, and it has succeeded. Execution time of this step of m-2 module has shut down from 18s on one processor...
5.2. Parallelization of message vector generation (last step)

The third step of the $m$-2 module consists in file reading, word comparison (through hash-tables), and in message vectors building and saving. As for first step of $m$-I module, file sizes and processing times are different, and we have chosen to parallelize these operations with a dynamic load balancing strategy. Each thread processes one file, and when it finishes it asks for the next file still unprocessed. But Next-File-Identification and Vector-Base-Writing operations are critical sections that limit parallelism (see figure 5.4). We reach interesting speed up only when the vector size is not too large, i.e. when user chose a not too great number of relevant words. See next section for performance measurements.

We have improved the dynamic load balancing mechanism of $m$-2 module, sorting files containing the filtered messages of each category (the tasks to process) from the longest to process to the shortest. Then each thread begins processing a long file, and finishes processing a short one. This improvement was motivated by the light number of files to process, i.e. by the light number of tasks to run. This experimental improvement has been successfully tested with processing time estimated from the number of messages in the category. It has led to less execution time than basic dynamic load balancing.

5.3. $m$-2 module global performances

As for $m$-I module, our parallelization contains many synchronization barriers, critical sections, unperfect load balancing, a small serial part and many disk accesses on basic PC architecture. But table 2 shows that we finally reach speed up.

However, significant speed up is reached only using a medium number of relevant words, leading to not too long message vectors and to not
too long files to write. Then speed up can reach 2.7 on four processors. When using large number of relevant words, speed up is negligible.

6. Parallelization of two vector classification algorithms

Many classification algorithms can be used to classify messages. Our Document-Categorization application offers three possibilities, and we have parallelized two: the basic algorithm of the K-nearest-neighbors (Cover et al., 1967), and a fast stochastic algorithm named Naïve-Bayes (McCallum et al., 1998).

6.1. Parallelization of the K-nearest-neighbors algorithm

This basic classification algorithm (K-nn) is slow and not very good, but is frequently used by our user-colleagues as a performance reference in text classification. So, they frequently run this slow algorithm and were very interested to speed up it. It is based on computation of the distance between the vector of each message to classify, and vectors of messages from a learning base. Then the algorithm looks for the K nearest messages of the learning base, and the most present category in this set of K documents is affected to the message under test. In our application of classification algorithm evaluation, the learning base and the test base come both from the initial message database. A part is used as learning database and the other as test database, the ratio is fixed by the user.

This third module contains many disk accesses to vector files, but all vector files are binary files (not ascii ones) and vector read operations are fast. We have chosen to not parallelize these disk accesses, but to focus on textual distance computation and on K-nearest-neighbors identification. The first step of this module creates two compacted databases (learning and test ones): it reads vector files, suppresses null components, and writes results in shorter binary files. This step has been parallelized on just two processors, using one processor for learning database, and another for test database. Then learning and test databases are read from disk and stored in memory per block, in order to be able to process very large databases. All these block reading are sequential,
but distance computation of test vectors to all learning vector are parallelized. Each thread processes independently an equal size part of the current block of test vectors that is fixed at the beginning of the parallel computation: this is data-partitioning. But computation times depend on the number of non-zero components of vectors. So load balancing is unperfect and is statistical load balancing: we assume that each part of block of test vectors contains approximately the same total amount of computation.

There is no critical section in this parallelization, and it reaches good speed up: up to 3.4 on four processors for a large database. See table 3 for more details on performances of m-3 module with K-nn algorithm.

6.2. Parallelization of a stochastic algorithm

Another interesting classification is based on a stochastic algorithm: Naive-Bayes algorithm (McCallum et al., 1998). It is very fast and efficient, and its parallelization is straightforward. Some stochastic factors are computed when identifying the relevant words (in m-2 module), and then each vector enters a stochastic computation loop, that finishes pointing out its category. All vector processing are independent, and are
parallelized with a data-partitioning strategy: each thread processes the same number of vectors that are fixed at the beginning of the parallel computation. As processing times are not exactly identical, this is again statistical load balancing.

<table>
<thead>
<tr>
<th>Number of proc.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. time (s)</td>
<td>0.25</td>
<td>0.23</td>
<td>0.23</td>
<td>0.28</td>
</tr>
<tr>
<td>Speed Up</td>
<td>1.0</td>
<td>1.1</td>
<td>1.1</td>
<td>0.9</td>
</tr>
<tr>
<td>Efficiency (%)</td>
<td>100</td>
<td>54</td>
<td>36</td>
<td>22</td>
</tr>
</tbody>
</table>

Large database & 200 relevant words

<table>
<thead>
<tr>
<th>Number of proc.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec. time (s)</td>
<td>2.87</td>
<td>2.58</td>
<td>2.48</td>
<td>2.46</td>
</tr>
<tr>
<td>Speed Up</td>
<td>1.0</td>
<td>1.1</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Efficiency (%)</td>
<td>100</td>
<td>56</td>
<td>39</td>
<td>29</td>
</tr>
</tbody>
</table>

Table 4: **Performance of the entire third module: message classification with Naïve-Bayes stochastic algorithm, on a 4xPII-450MHz PC with SCSI disk technology**

No critical section and very few synchronization barriers are present in this parallelization. But file reading has been remained sequential (as for K-nn algorithm parallelization), because we have never succeeded parallelizing fast binary file accesses on shared memory multiprocessor-PC architecture with classical disk technology. Finally, speed up exists but is poor because parallel computations are very fast and main time is spent in serial disk accesses. See table 4 for details. Anyway, this stochastic algorithm leads to very short execution times for m-3 module. So, its poor speed up does not have impact on global parallelization performance of the entire application, and this last parallelization effort has appeared not profitable on low cost shared memory multiprocessor PC. However, it could be profitable on architectures with parallel file accesses (such as multiprocessor PC with RAID disk technology), but we have not done any test at this time.
7. Global tests and performances of the application

![Execution times of the entire application on a large database of 20000 messages (top) and on a medium 1872 message database (bottom), with 200 (left) and 2000 (right) relevant words, for two different classification algorithms on a 4xPII-450MHz with SCSI disk technology.](image)

Figure 7.5: Execution times of the entire application on a large database of 20000 messages (top) and on a medium 1872 message database (bottom), with 200 (left) and 2000 (right) relevant words, for two different classification algorithms on a 4xPII-450MHz with SCSI disk technology.
Figure 7.6: Speed Up of the entire application on a large database of 20000 messages (top) and on a medium 1872 message database (bottom), with 200 (left) and 2000 (right) relevant words, for two different classification algorithms on a 4xPPI-450MHz with SCSI disk technology.

7.1. Parallel performances on a shared memory 4-processor PC

We have evaluated the parallelization of our Document-Categorization application on our medium database: 1872 messages, and on our large database: 20000 messages, and we have used 200 and 2000 relevant words to encode messages leading to message vectors of 200 and 2000 components. Moreover, each test has been made with \( K\)-nearest-neighbors and with \( \text{Naïve-Bayes} \) classification algorithm.
Figure 7.5 and 7.6 introduce execution times and speed up measured on a 4xPII-450MHz PC with SCSI disk technology. We observe that:

- Classification algorithm used in \( m \)-3 module has sensitive impact on final execution time and on speed up of the entire application, mainly when categorizing large databases.
- Speed up reaches 2.2 to 2.5 on 4 processors when processing our medium or large database with a small number of relevant words (short message vector files), and reaches only 1.7 to 2.4 when using a high number of relevant words (long message vector files). So, when algorithm is fixed the most relevant speed up parameter appears not to be the size of the database, but number of words used to encode the messages.
- Best speed up are always reached on 4 processors. Sometimes speed up on 3 and 4 processors are close, but always sensitively better on 4 processors than on 2 processors. So, use of a 4-processor PC is really most interesting than use of a 2-processor PC.

As expected: unperfect load balancing, waiting times on critical sections or serial execution of concurrent IO operations, limit parallelization performances. However, complete application reaches speed up with efficiency close to 60% when using small number of relevant words to classify documents. When using more relevant words, efficiency of the parallelization on 4-processor PC shuts down under 50% (excepted for \( K-nn \) algorithm on large database). Then it could be more interesting to limit parallelization to 2-processor PCs and to reach efficiency greater than 70%.

Finally, we are fairly satisfied of these performances reached by our first entire parallelization of this application, on low-cost shared memory PC with up to 4 processors and Windows OS.

### 7.2. Impact of the disk technology

As parallelization on low cost multiprocessor-PC allows easy change of parallel computer, we have experimented our parallelization on a more recent 2xPIII-700MHz PC, but with just IDE disk technology. Table 5 shows these performances on our large database. We can observe that execution times are greater on this machine than on 2 processors.
of our 4xPIII-450MHz PC with SCSI technology. A fine analysis of the
different module execution times shows that slow down is focussed on
\textit{m-1} module: module with long \textit{ascii} file accesses. This phenomenon
illustrates that disk access technology is as important as processor tech-
nology for our complete application of document categorization, and for
its parallelization. It is not interesting to spent all financial resources in
processors, but both in processors and disk accesses.

Finally, we reach longer execution times, but good speed up with ef-
ficiency from 86\% to 89\%, close to the efficiency on 2 processors of our
4xPIII-450MHz PC with SCSI disk (78\% to 87\%). So, if we can not buy
PC with good disk technology but only poor PC, then parallelization on
poor 2-processor shared memory PC remains interesting.

Anyway, as multiprocessor PCs are cheap we can easily run our par-
allelization each year on a new parallel computer, with identical disk
technology but with new processors, and for approximately the same
price. Then we can easily continue to run faster than on new sequen-
tial PC. This is hard to achieve on expensive high performance parallel
machines, because we can not upgrade these machines frequently! We
think it is important to obtain \textit{constant} speed up at \textit{constant} price when
technology evolves, that leads to permanent advance on sequential tech-
nology (see (Vialle, 2002)). Design efficient parallelization on low cost
parallel computer is a way to reach this goal.
Figure 8.7: Example of a categorization application run under Windows and on a 4-processor shared memory PC. We can observe the task manager of Windows showing the four processors working simultaneously from the beginning of the run.

8. Conclusion and Perspectives

8.1. Present speed of document categorization

Due to our parallelization, we are able to process 34.0 to 57.1 test documents per second of our large database: speed to process test messages and all learning messages of the 20000 message database. These results correspond to fairly satisfying speed for the entire application, including many disk accesses, on our low cost 4-processor PC. Figure 8.7 and 8.8 are screen-shots of a parallel run under Windows, with the task manager showing simultaneously beginning and end of the four processor activity. But performances remain insufficient to process larger database and permanent flow of new data, such as Internet news groups.

So, this research was just a first step, to test parallelization potential of this application on low-cost shared memory multiprocessor PC, and
Figure 8.8: End of a parallel run under Windows and on a 4-processor shared memory PC. The four processors end simultaneously, when all messages have been processed.
to identify profitable and unprofitable module parallelization. Now we have to look for more performance on still low cost processing systems, and we introduce several possible ways in the next sections.

8.2. Optimization of the current parallelization

We have identified three ways of evolution of this research in the future. The first way consists in optimizations of the parallel implementation and small modifications of the different algorithms, in order to obtain better speed up on shared memory multiprocessor PCs without any change in the application specifications. Some small parts of the application remain sequential, but it is not sure their parallelization would be profitable, needing concurrent IO technology (as for stochastic algorithm, section 6.2) or being insignificant part of the global execution time (Amdahl’s law (Amdahl, 1967) is not a strong limitation on just 4 processors). We prefer to investigate impact of shared memory and cache memory sizes and perhaps to make better usage of cache memories to reduce accesses to shared memory.

The second way would be to design a new application without intermediate file storing, in order to decrease the number of disk accesses that remain slow and do not support concurrency efficiently on PC architecture. This evolution would lead to change in the specification of the application, and depends on the needs of our user-colleagues.

The third way is about more efficient architecture use: some recent 2-processor PCs have two memory bus and seem to better support concurrent shared memory accesses, and some PC servers have efficient disk access mechanisms and could support concurrent file reading or writing. But high performance PC are sometimes expensive, and can become difficult to upgrade frequently (for money constraints). This way of evolution has to be studied from both technical and financial point of view. Our objective to use standard and cheap PCs remains.
8.3. Evolution to clusters of multiprocessor PCs

Speed up measured on our shared memory PC with up to 4 processors, is interesting and can be optimized, but it seems to indicate that performances of the current parallelization on shared memory PC begins to significantly loose efficiency when using 4 processors (see curves on figure 7.6). Try to use larger shared memory PCs seems not to be the solution to reach really larger speed up. A better solution could be to use a cluster of different small multiprocessor PCs, each with its own memory and memory bus, and its own disk and disk access bus.

But to avoid to waste too many time in message passing, we need to design a parallelization scheme with good data partitioning and computation locality, without entire document vector exchanges between processors, with just some short variable exchanges. For example, (Agrawal et al., 1996) and (Foti et al., 2000) introduce parallelizations based just on some count exchanges, or on distributed reduce operations. But our main approach is to adopt a multi-stage categorization algorithm, with a first global and light categorization, and then fine and independent categorizations inside category subsets that should be run concurrently on nodes of a multiprocessor PC cluster.

Low cost clusters of multiprocessor PCs can be upgraded frequently and can reach very good speed up with adapted parallel algorithms, it means cumulating as many parallelization strategies as needed. In the future we wish to experiment this way on Document-Categorization applications, and on more complete Text-Mining applications.

Acknowledgment

This research was supported by Region Lorraine (France), and the multiprocessor PC used has been issued with the help of Charles Hermite Center (France), Conseil Général de la Moselle and Mairie de Metz.

Authors want to thank Yan-Eric Douvier (Supélec student) for its contribution to parallel developments, and Gilles Berlin (Supélec technical staff) for its help to parallel computer configuration.
A. References

Agrawal R., Shafer J., "Parallel Mining of Association Rules", IEEE Transactions on Knowledge and Data Engineering, December, 1996.


Predicates for state changes vs processes for event patterns
Prédicats de changements d’état vs processus
Tommaso Bolognesi
C.N.R. - ISTI
1, Via Moruzzi, 56124
Pisa, Italy
t.bolognesi@isti.cnr.it